

Interactive Metaprogramming Shell Based on Clang

Ábel Sinkovics, András Kucsma

Agenda

- What is template metaprogramming?
- Where is TMP used?
- How TMP works?
- How to do TMP with Metashell?
- Demos

Template metaprogramming

What is template metaprogramming?

Template metaprogramming

- "Disaster" Alex Ispánovics
- "Executing compile-time algorithms with the help of cleverly defined templates" Zoli<1>
- "It is not something people can not deal with, it is something compilers can not deal with" Tibi
- "Unwaitable compile-time and infinite memory needs" Dani
- "Intellectual entertainment" Zoli<2>

Template metaprogramming

- "Disaster" Alex Ispánovics
- "Executing compile-time algorithms with the help of cleverly defined templates" Zoli<1>
- "It is not something people can not deal with, it is something compilers can not deal with" Tibi
- "Unwaitable compile-time and infinite memory needs" Dani
- "Intellectual entertainment" Zoli<2>

He is a professor!!!

min

```
int main()  
{  
    std::cout << min(2, 3 ) << std::endl;  
}
```

min

```
int main()  
{  
    std::cout << min(2, 3 ) << std::endl;  
}
```

```
template < class T > min(T a, T b)  
{  
    return a < b ? a : b;  
}
```

min

```
int main()  
{  
    std::cout << min(2, 3.4) << std::endl;  
}
```

```
template < class T > min(T a, T b)  
{  
    return a < b ? a : b;  
}
```


min

```
int main()  
{  
    std::cout << min(2, 3.4) << std::endl;  
}
```

```
template <class A, class B> min(A a, B b)  
{  
    return a < b ? a : b;  
}
```

min

```
int main()  
{  
    std::cout << min(2, 3.4) << std::endl;  
}
```



```
template <class A, class B>                min(A a, B b)  
{  
    return a < b ? a : b;  
}
```

min

```
int main()
{
    std::cout << min(2, 3.4) <<
}
```

```
template <class A, class B>
struct common_type
{
    typedef /* ... */ type;
}
```



```
template <class A, class B>
    min(A a, B b)
{
    return a < b ? a : b;
}
```

min

```
int main()
{
    std::cout << min(2, 3.4) <<
}
```

```
template <class A, class B>
struct common_type
{
    typedef /* ... */ type;
}
```

```
template <class A, class B>
typename common_type<A, B>::type min(A a, B b)
{
    return a < b ? a : b; // ...
}
```

min

```
template <class A, class B>  
struct common_type;
```

min

```
template <class A, class B>
struct common_type;

template <>
struct common_type<int, double>
{
    typedef double type;
};
```

min

```
template <class A, class B>
struct common_type;

template <>
struct common_type<int, double>
{
    typedef double type;
};

// ...
```

min

```
template <class A, class B>  
struct common_type;
```

```
template <>  
struct common_type<int, double>  
{  
    typedef double type;  
};
```

```
// ...
```

```
template <class T>  
struct common_type<T, T>  
{  
    typedef T type;  
};
```


min

```
template <class A, class B>
struct common_type
{
    using type =
        decltype(true ? std::declval<A>() : std::declval<B>());
};
```

```
template <class... Ts>
struct common_type
{
    // ...
};
```

Metaprogramming in C++

C++ source code



Metaprogram

Metaprogramming in C++

C++ source code

Metaprogram

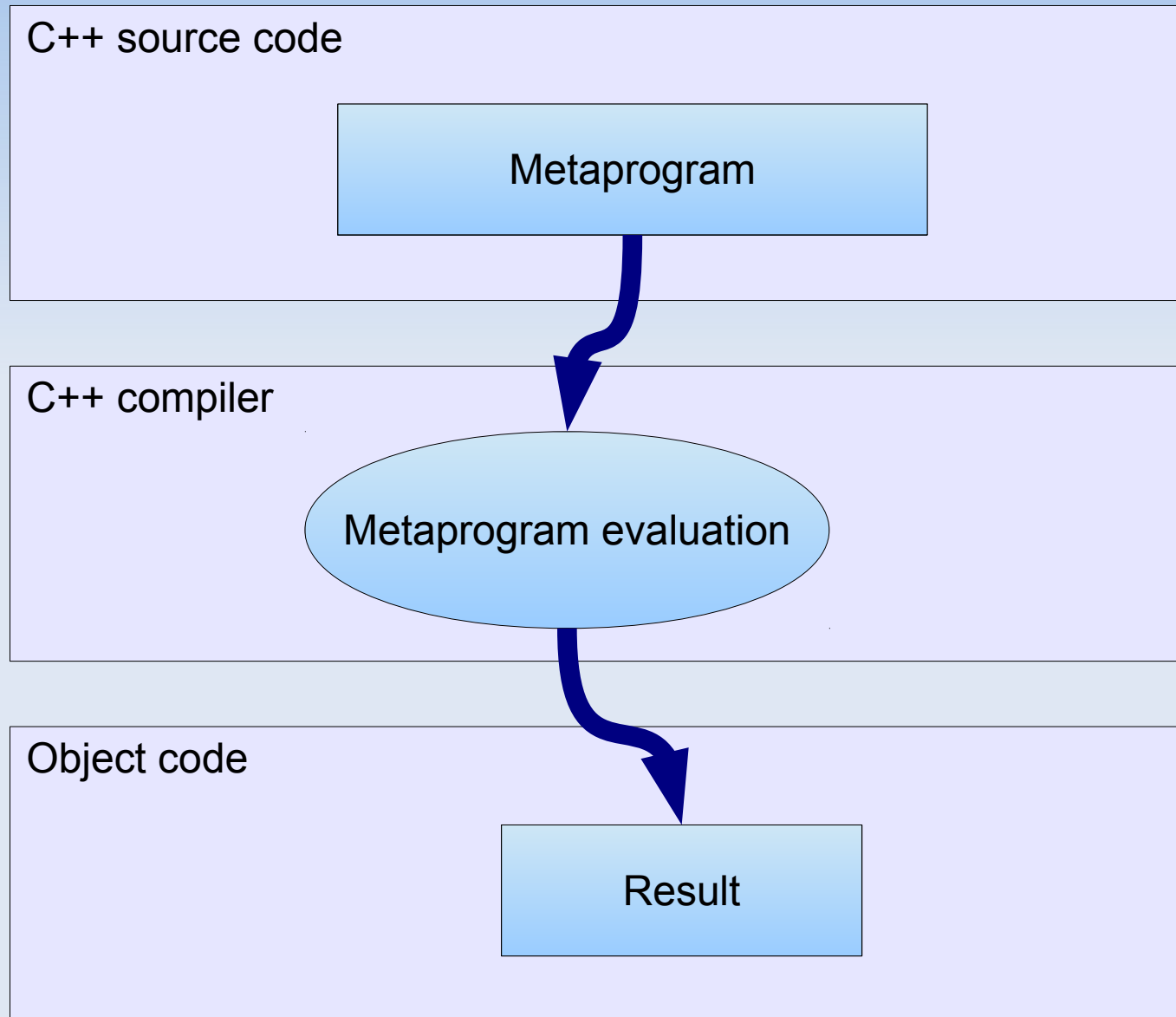
C++ compiler

Metaprogram evaluation

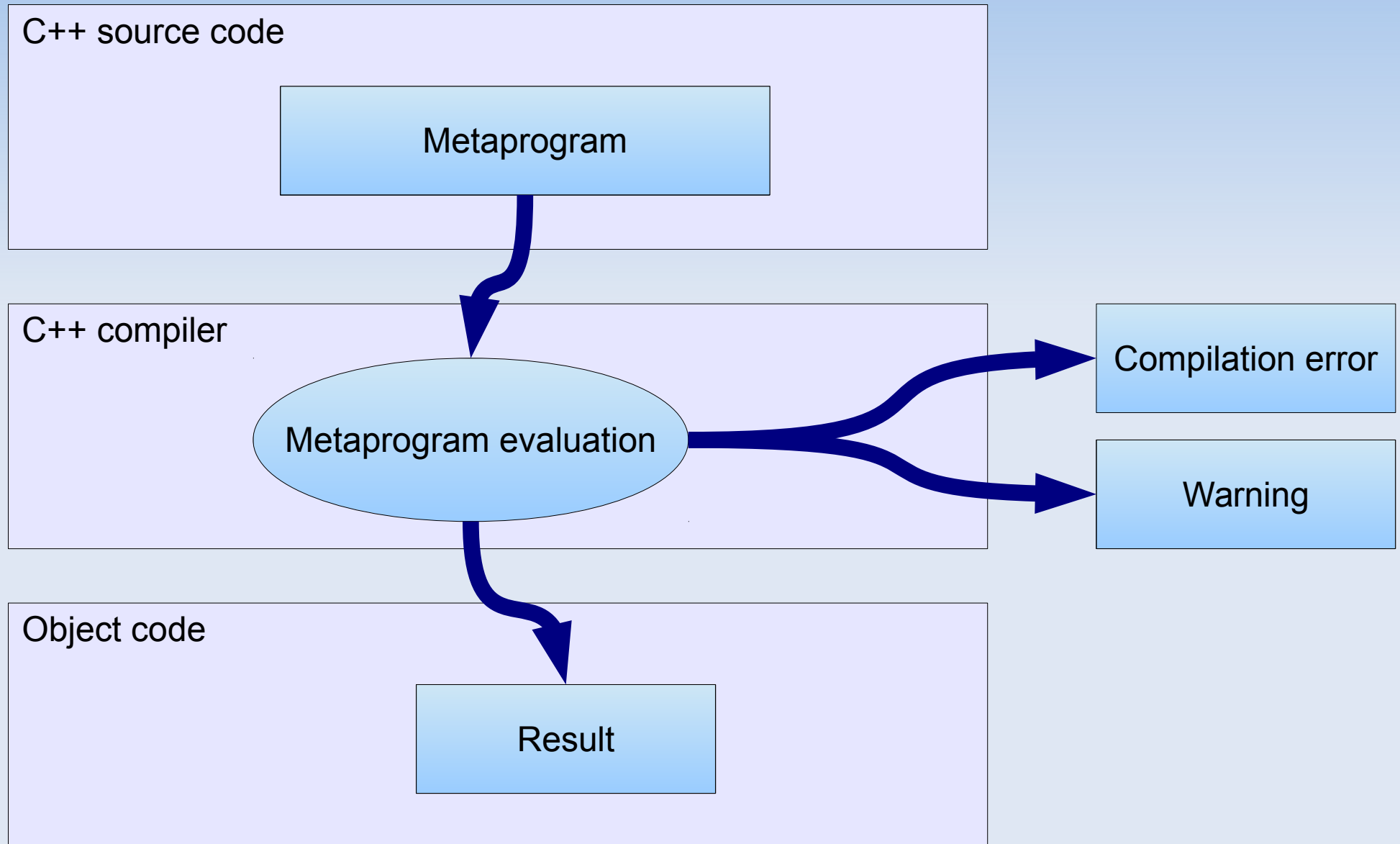
```
graph TD; subgraph SourceCode [C++ source code]; direction TB; MP[Metaprogram]; end; subgraph Compiler [C++ compiler]; direction TB; MPE([Metaprogram evaluation]); end; MP --> MPE;
```

The diagram illustrates the process of metaprogramming in C++. It is divided into two main sections: 'C++ source code' and 'C++ compiler'. In the 'C++ source code' section, a light blue rectangular box labeled 'Metaprogram' is centered. A thick blue arrow originates from the bottom of this box and points downwards towards the 'C++ compiler' section. In the 'C++ compiler' section, a light blue oval labeled 'Metaprogram evaluation' is centered, representing the execution of the metaprogram within the compiler's environment.

Metaprogramming in C++



Metaprogramming in C++



Application areas

- Type calculations in generic code
- Simulating/experimenting with language features
- Extra "optimisations"
- Validation
- Domain specific language embedding

Lambda expressions

- Lambda expressions in C++98 (Boost.Phoenix)

Lambda expressions

- Lambda expressions in C++98 (Boost.Phoenix)
- Display the values greater than 5
`std::vector<int> v;`

Lambda expressions

- Lambda expressions in C++98 (Boost.Phoenix)

- Display the values greater than 5

```
std::vector<int> v;  
std::for_each(v.begin(), v.end(),
```

```
);
```

Lambda expressions

- Lambda expressions in C++98 (Boost.Phoenix)

- Display the values greater than 5

```
std::vector<int> v;  
std::for_each(v.begin(), v.end(),  
    if_(  
  
    )  
  
);
```

Lambda expressions

- Lambda expressions in C++98 (Boost.Phoenix)

- Display the values greater than 5

```
std::vector<int> v;  
std::for_each(v.begin(), v.end(),  
    if_(arg1 > 5)
```

```
);
```

Lambda expressions

- Lambda expressions in C++98 (Boost.Phoenix)
- Display the values greater than 5

```
std::vector<int> v;  
std::for_each(v.begin(), v.end(),  
    if_(arg1 > 5) [  
  
    ]  
);
```

Lambda expressions

- Lambda expressions in C++98 (Boost.Phoenix)

- Display the values greater than 5

```
std::vector<int> v;  
std::for_each(v.begin(), v.end(),  
    if_(arg1 > 5) [  
        std::cout << arg1 << ", "  
    ]  
);
```

Expression templates

- Todd Veldhuizen
- Do "optimisation"s at compile-time
- Extra "optimisation"s based on domain knowledge
 - The compiler is not aware of them

Parsing at compile time

- Take a string literal and turn it into a C++ construct (value, object, function, template metaprogram, etc.)

Parsing at compile time

- Take a string literal and turn it into a C++ construct (value, object, function, template metaprogram, etc.)
- Examples:
 - `checked_query("select LOCATION from EMPLOYEE join DEPARTMENT where DIVNUM = 2")`
 - `checked_regex("x[ab]")`
 - `RULE("S ::= DOUBLE (' , ' DOUBLE)*")`

Parsing at compile time

- Take a string literal and turn it into a C++ construct (value, object, function, template metaprogram, etc.)
- Examples:
 - `checked_query("select LOCATION from EMPLOYEE join DEPARTMENT where DIVNUM = 2")`
 - `checked_regex("x[ab]")`
 - `RULE("S ::= DOUBLE (' , ' DOUBLE)*")`
- Constexpr-based solution: Sprout
- TMP-based solution: Mpllibs.Metaparse

C++ template metaprogramming

```
template <int N>  
struct fact  
{  
  
};
```

C++ template metaprogramming

```
template <int N>
struct fact
{
    static const int value =

};
```

C++ template metaprogramming

```
template <int N>
struct fact
{
    static const int value =
        N * fact<N - 1>::value;
};
```

C++ template metaprogramming

```
template <int N>
struct fact
{
    static const int value =
        N * fact<N - 1>::value;
};
```

```
template <>
struct fact<0>
{
    static const int value =
        1;
};
```

C++ template metaprogramming

fact<3>::value

```
template <int N>
struct fact
{
    static const int value =
        N * fact<N - 1>::value;
};
```

```
template <>
struct fact<0>
{
    static const int value =
        1;
};
```

C++ template metaprogramming

fact<3>::value

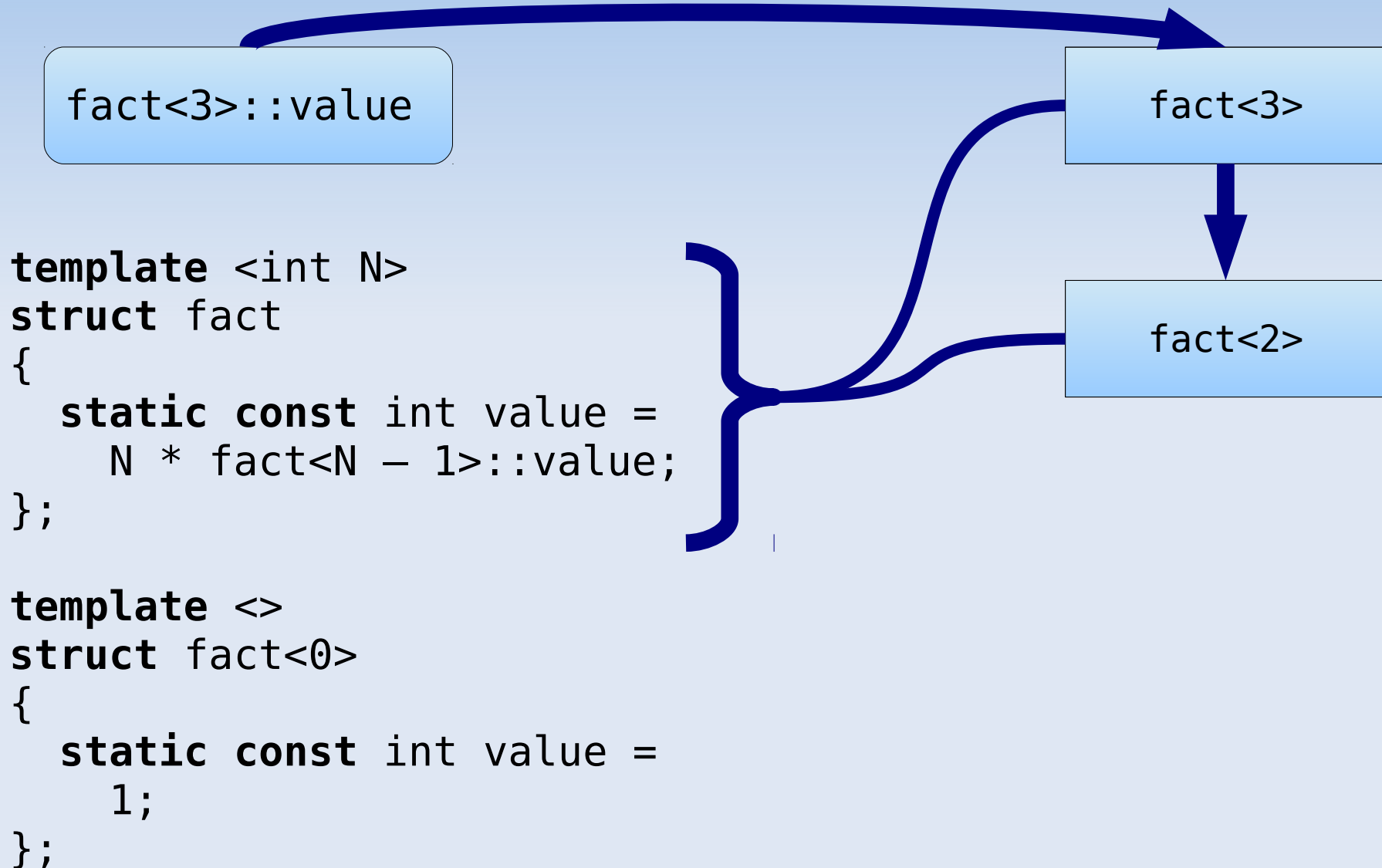
fact<3>

```
template <int N>  
struct fact  
{  
    static const int value =  
        N * fact<N - 1>::value;  
};
```

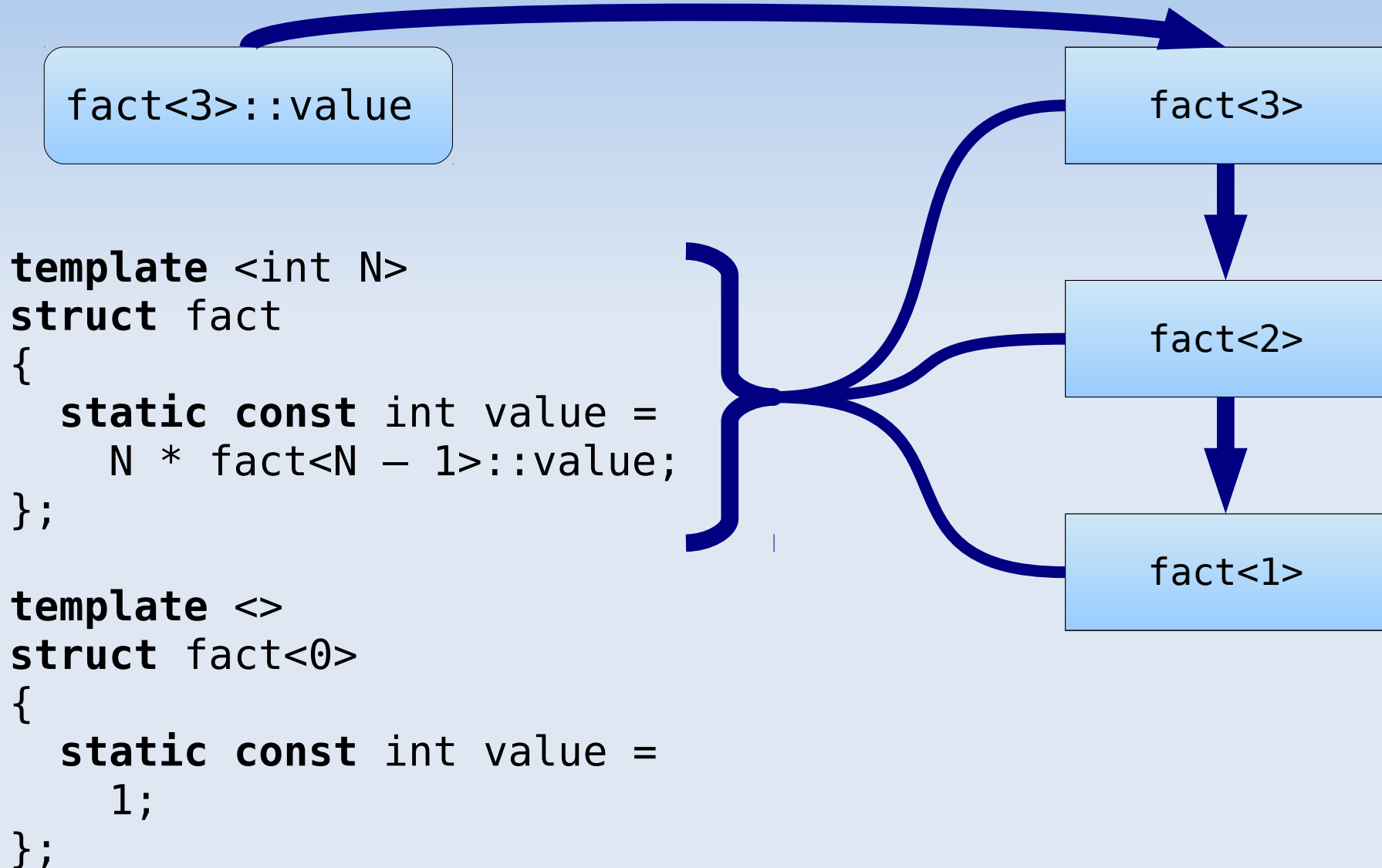
```
template <>  
struct fact<0>  
{  
    static const int value =  
        1;  
};
```



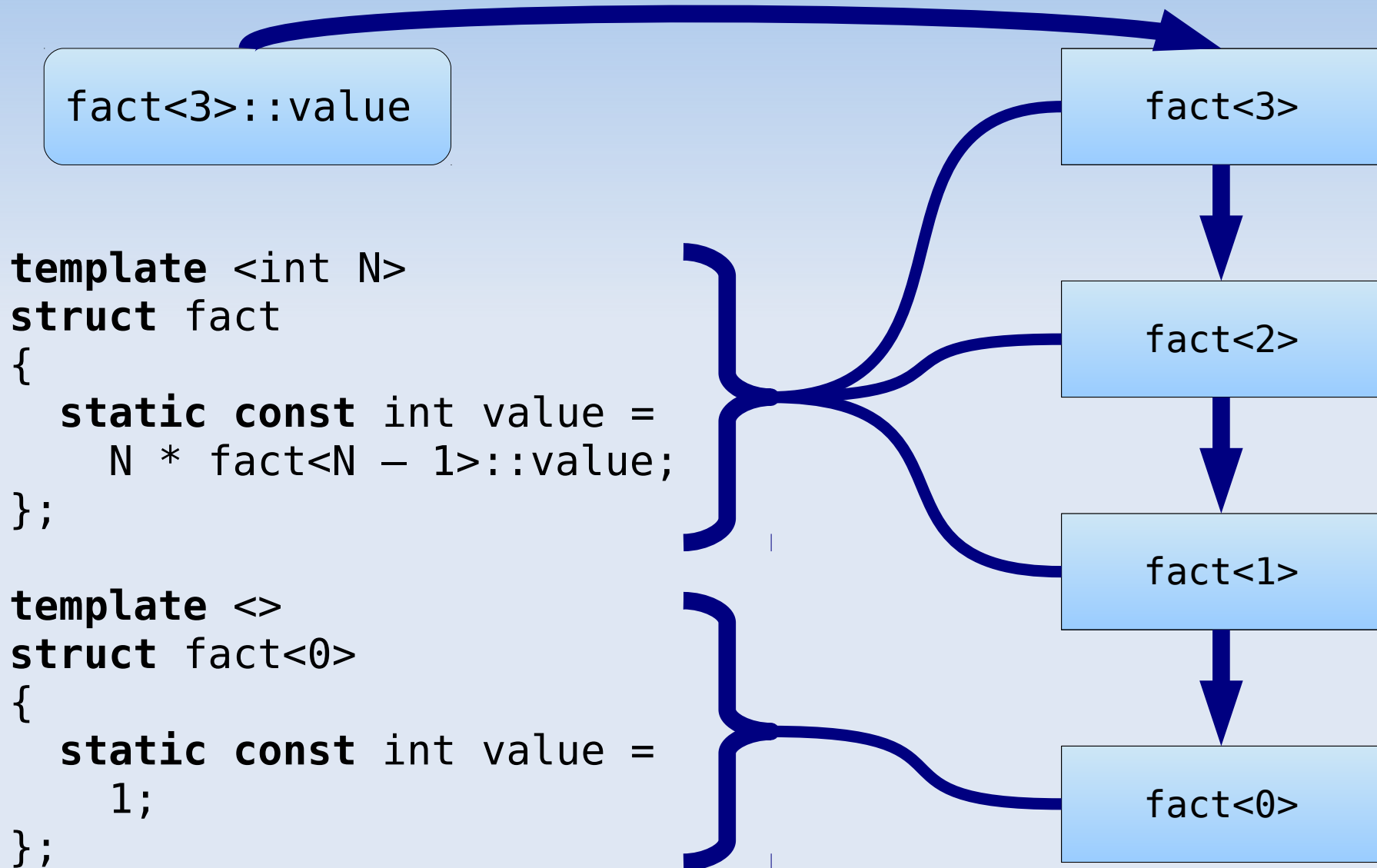
C++ template metaprogramming



C++ template metaprogramming



C++ template metaprogramming



C++ template metafunction

Argument list

Name

Body

C++ template metafunction

```
template <class T>  
struct add_const  
{  
    typedef const T type;  
};
```

Argument list

Name

Body

C++ template metafunction

```
template <class T>  
struct add_const  
{  
    typedef const T type;  
};
```

Argument list

Name

Body

C++ template metafunction

```
template <class T>  
struct add_const  
{  
    typedef const T type;  
};
```

Argument list

Name

Body

add_const<int>::type

C++ template metafunction

```
template <class T>  
struct add_const  
{  
    typedef const T type;  
};
```

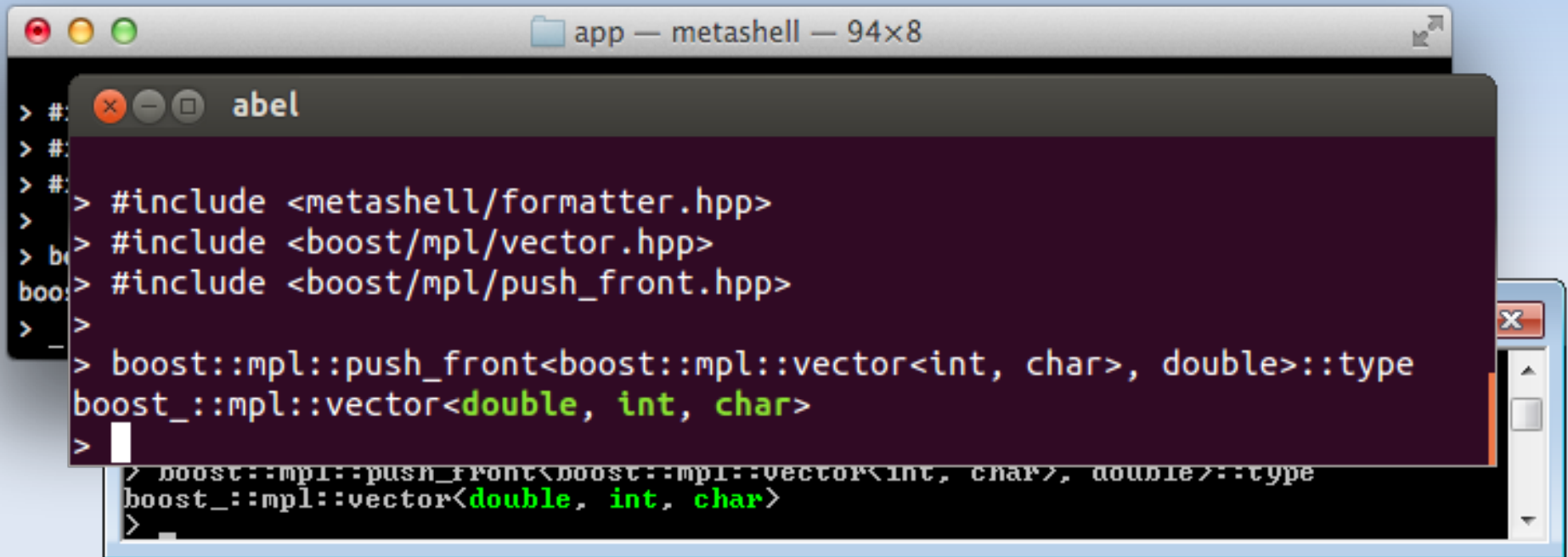
Argument list

Name

Body

add_const<int>::type

Metashell demo



The image shows a screenshot of a terminal window titled "app — metashell — 94x8". Inside the terminal, there is a sub-window titled "abel" with a dark background. The terminal displays the following C++ code:

```
> #include <metashell/formatter.hpp>
> #include <boost/mpl/vector.hpp>
> #include <boost/mpl/push_front.hpp>
>
> boost::mpl::push_front<boost::mpl::vector<int, char>, double>::type
boost_::mpl::vector<double, int, char>
>
```

The output line shows the result of the type deduction: `boost_::mpl::vector<double, int, char>`. The words `double`, `int`, and `char` are highlighted in green in the original image.

Metashell: <http://github.com/sabel83/metashell>

Online demo: <http://abel.web.elte.hu/shell>

Technical break

Tool support

Run-time

Compile-time

Tool support

Run-time

Compile-time



Tool support

Run-time



Compile-time

Tool support

Run-time



Compile-time

Tool support

Run-time



Compile-time



Tool support

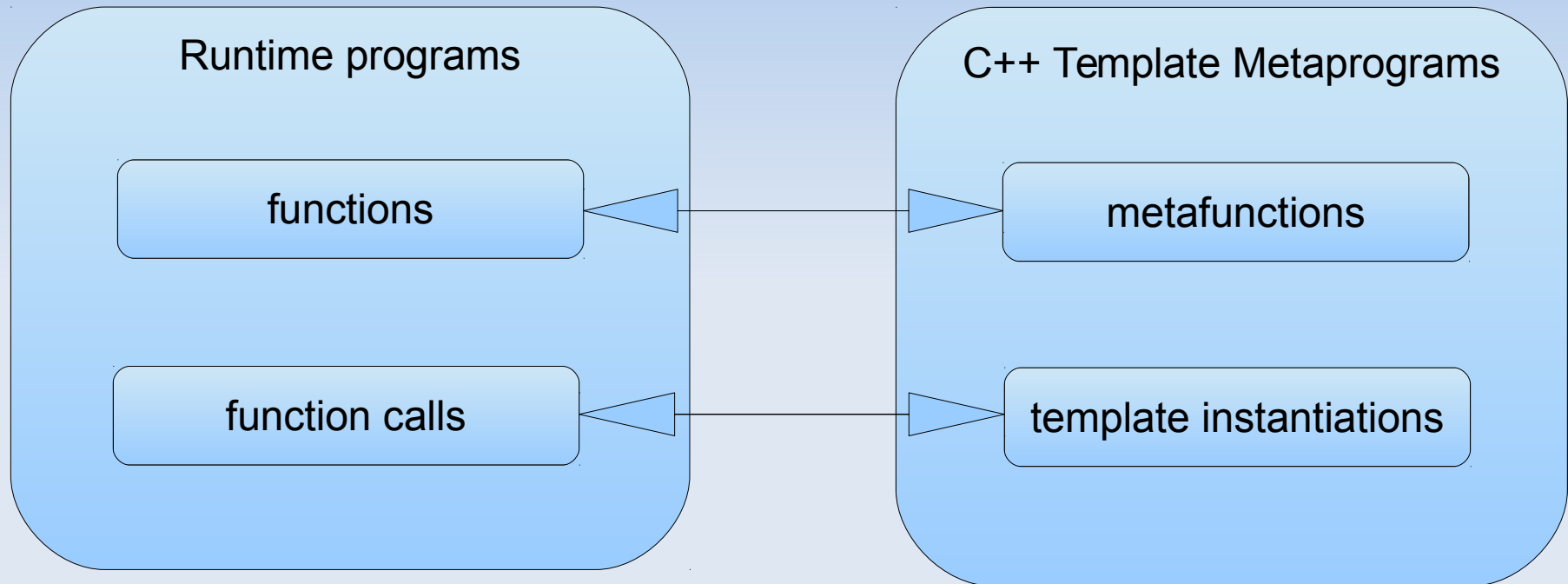
Run-time



Compile-time



Runtime- vs. Metaprograms



Fibonacci

```
template <class N>
struct fib :
    eval_if<
        typename
        less<
            N, int_<2>
        >::type,
        int_<1>,
        fib_impl<N>
    >::type
{};
```

Fibonacci

fib<int_<4>>::type

```
template <class N>
struct fib :
    eval_if<
        typename
        less<
            N, int_<2>
        >::type,
        int_<1>,
        fib_impl<N>
    >::type
{};
```

Fibonacci

fib<int_<4>>::type

fib<int_<4>>

```
template <class N>
struct fib :
    eval_if<
        typename
        less<
            N, int_<2>
        >::type,
        int_<1>,
        fib_impl<N>
    >::type
{};
```

Fibonacci

fib<int_<4>>::type

fib<int_<4>>

less<int_<4>, int_<2> >

```
template <class A, class B>
struct less :
    bool_<
        (A::value < B::value)
    >
{};
```

Fibonacci

fib<int_<4>>::type

fib<int_<4>>

```
template <class A, class B>  
struct less :  
    bool_<  
        (A::value < B::value)  
    >  
{};
```

less<int_<4>, int_<2> >

int_<4>

Fibonacci

fib<int_<4>>::type

fib<int_<4>>

```
template <class A, class B>  
struct less :  
    bool_<  
        (A::value < B::value)  
    >  
{};
```

less<int_<4>, int_<2> >

int_<4>

int_<2>

Fibonacci

fib<int_<4>>::type

fib<int_<4>>

```
template <class A, class B>  
struct less :  
    bool_<  
        (A::value < B::value)  
    >  
{};
```

less<int_<4>, int_<2> >

int_<4>

bool_<false>

int_<2>

Fibonacci

fib<int_<4>>::type

fib<int_<4>>

```
template <class N>
struct fib :
    eval_if<
        typename
        less<
            N, int_<2>
        >::type,
        int_<1>,
        fib_impl<N>
    >::type
{};
```

less<int_<4>, int_<2> >

int_<4>

bool_<false>

int_<2>

Fibonacci

fib<int_<4>>::type

```
template <class C,  
          class T, class F>  
struct eval_if :  
    if_c<  
        C::value, T, F  
    >::type  
{};
```

fib<int_<4>>

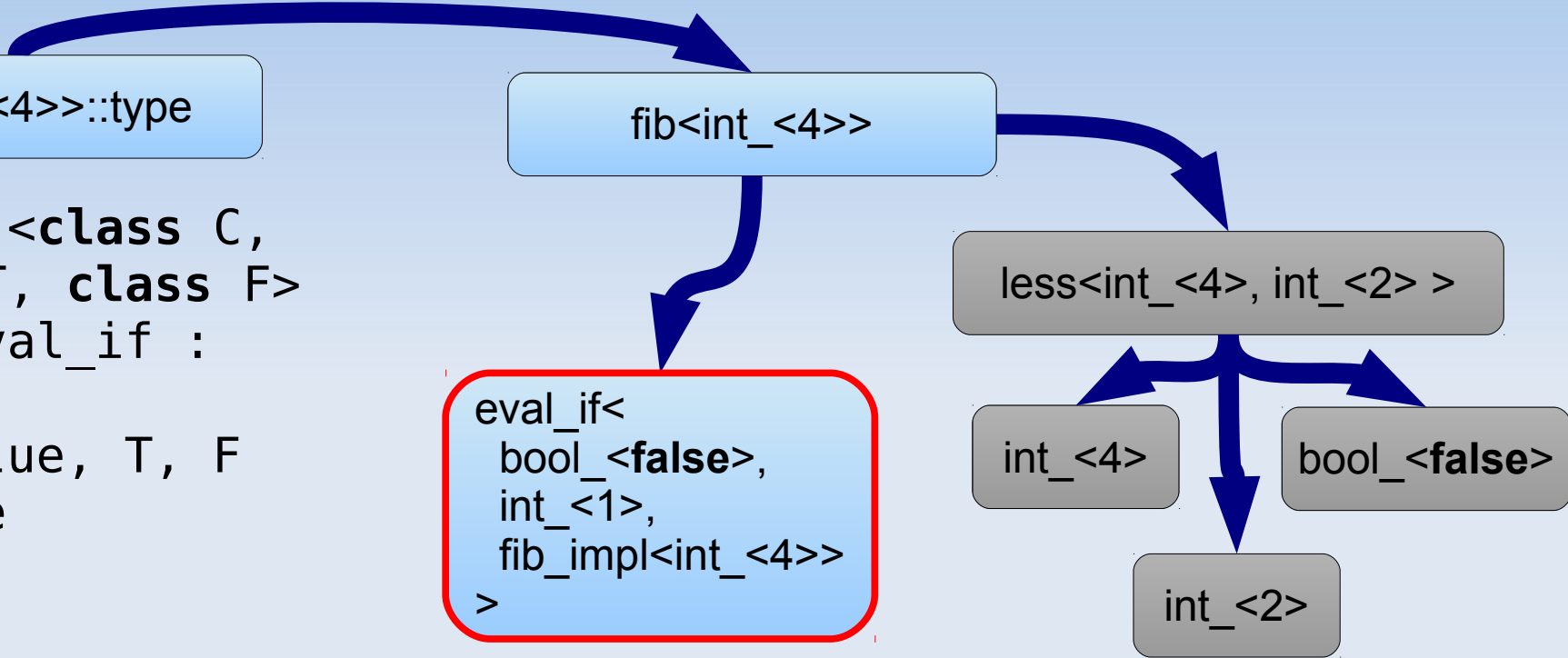
eval_if<
 bool_<false>,
 int_<1>,
 fib_impl<int_<4>>
>

less<int_<4>, int_<2> >

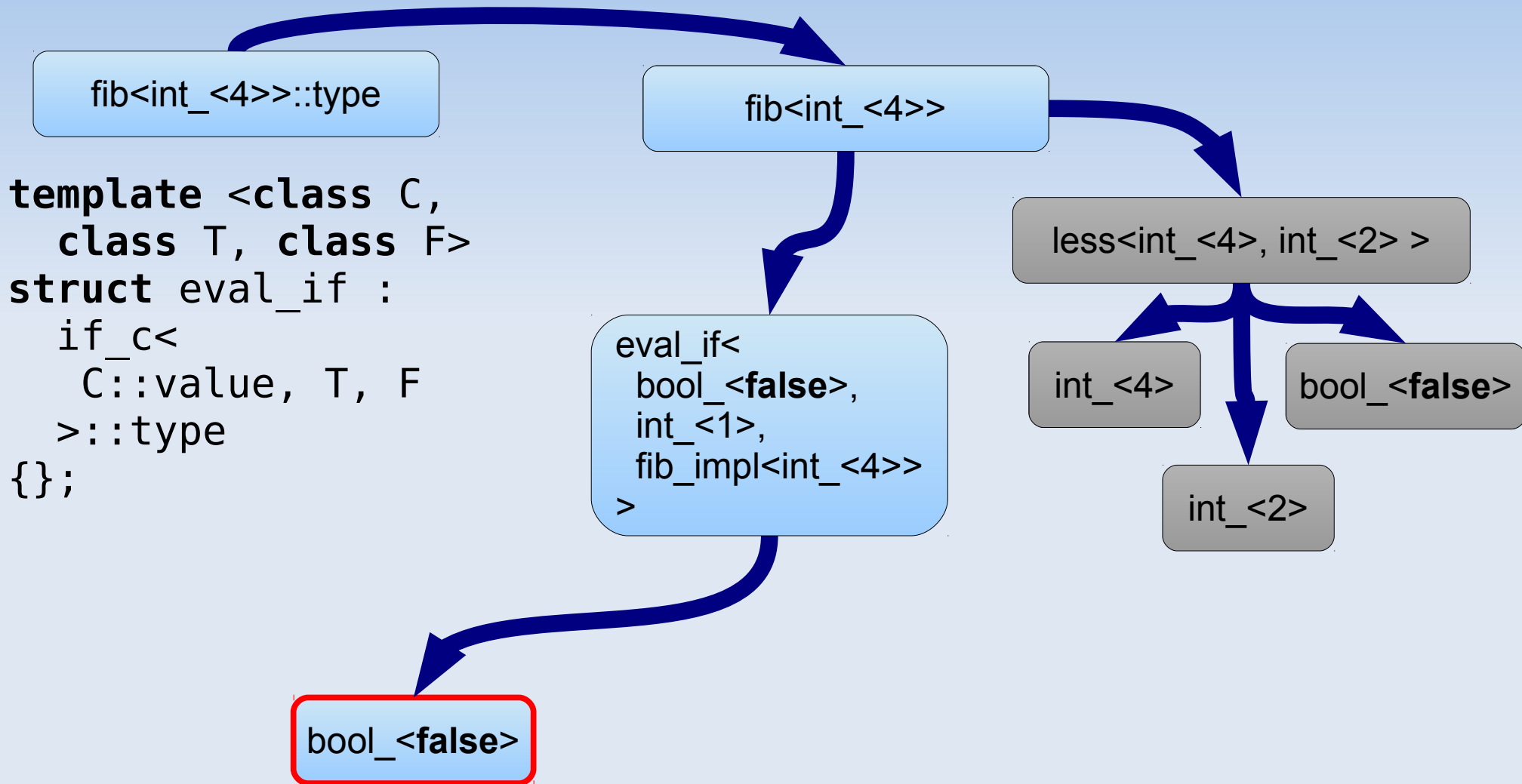
int_<4>

bool_<false>

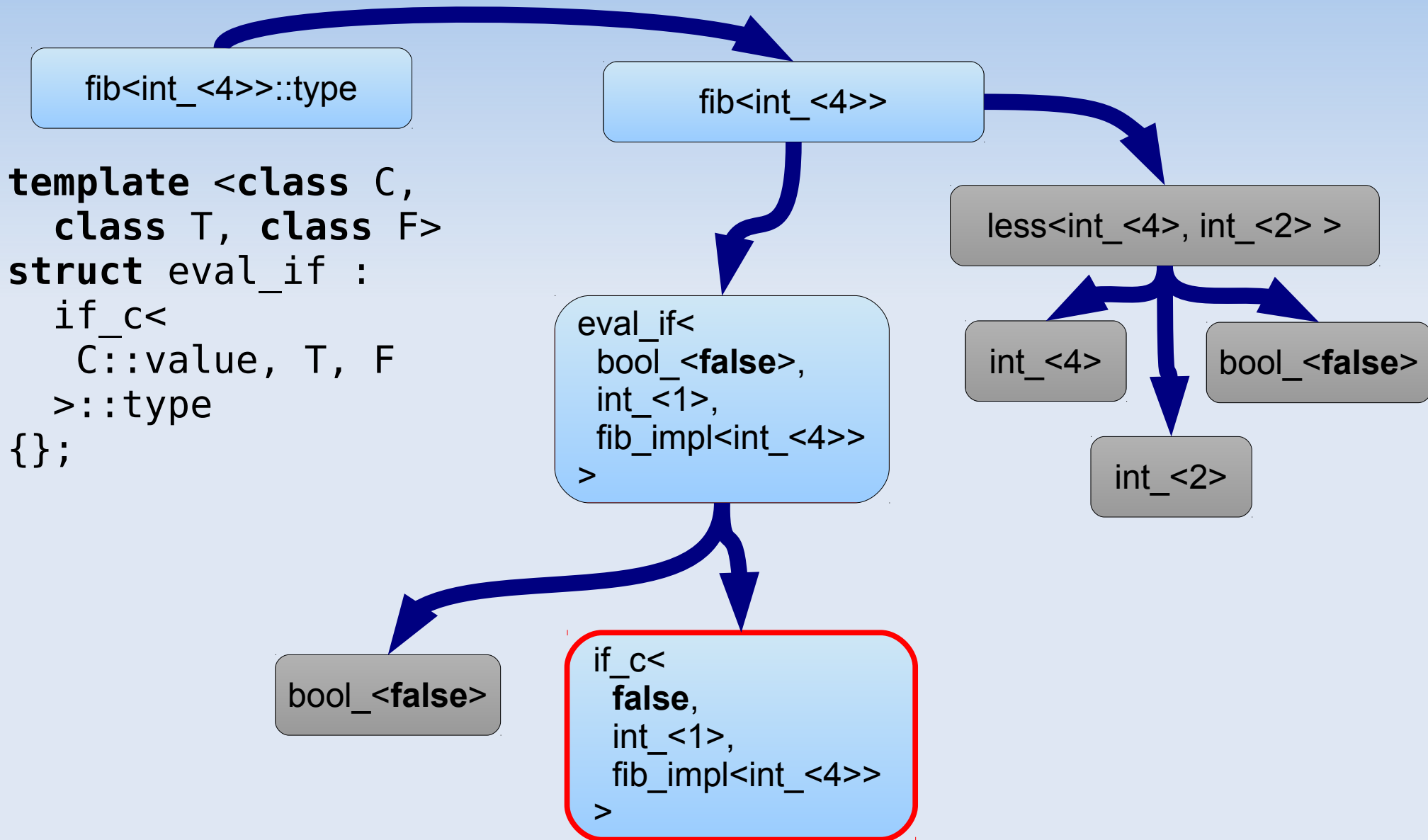
int_<2>



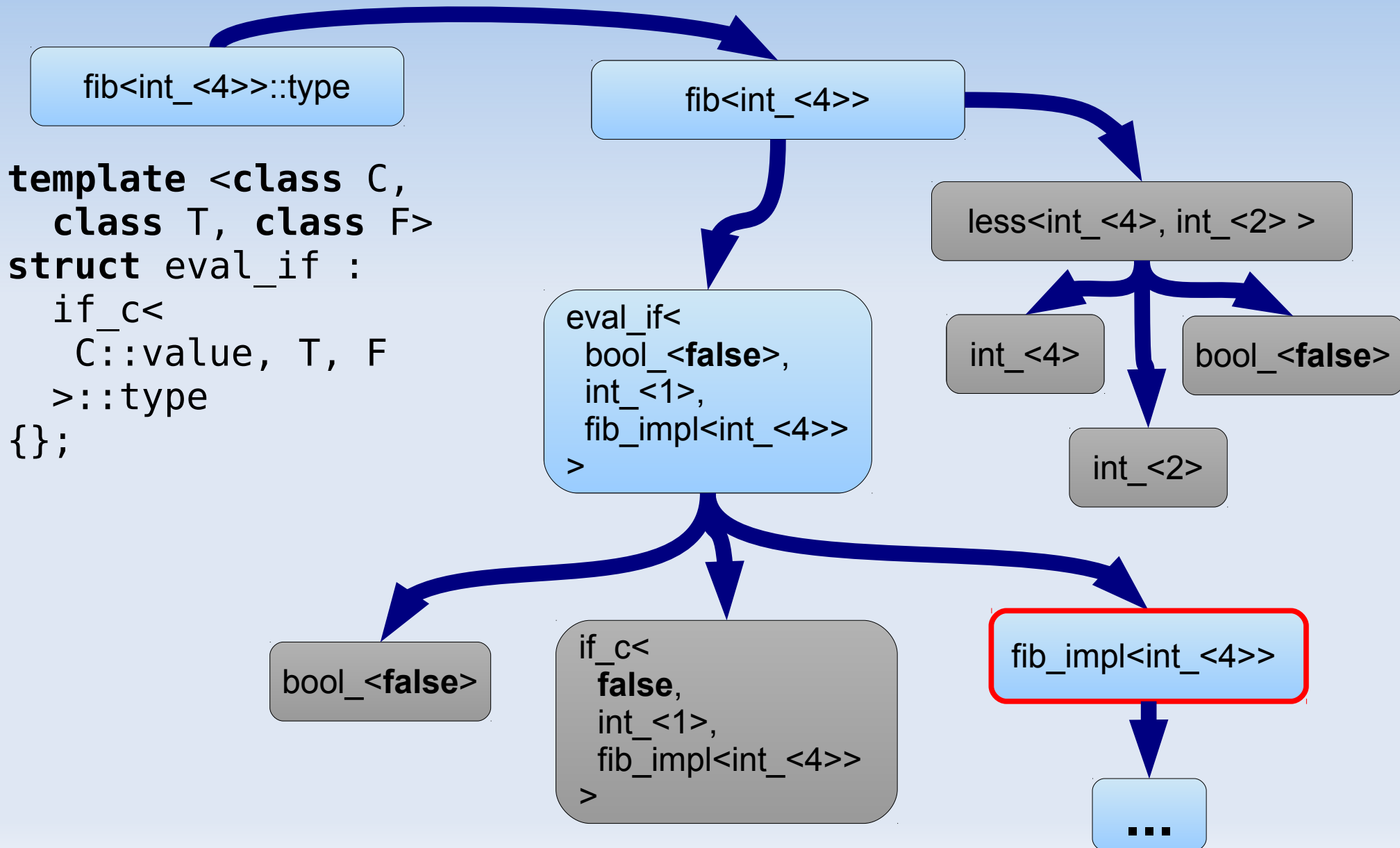
Fibonacci



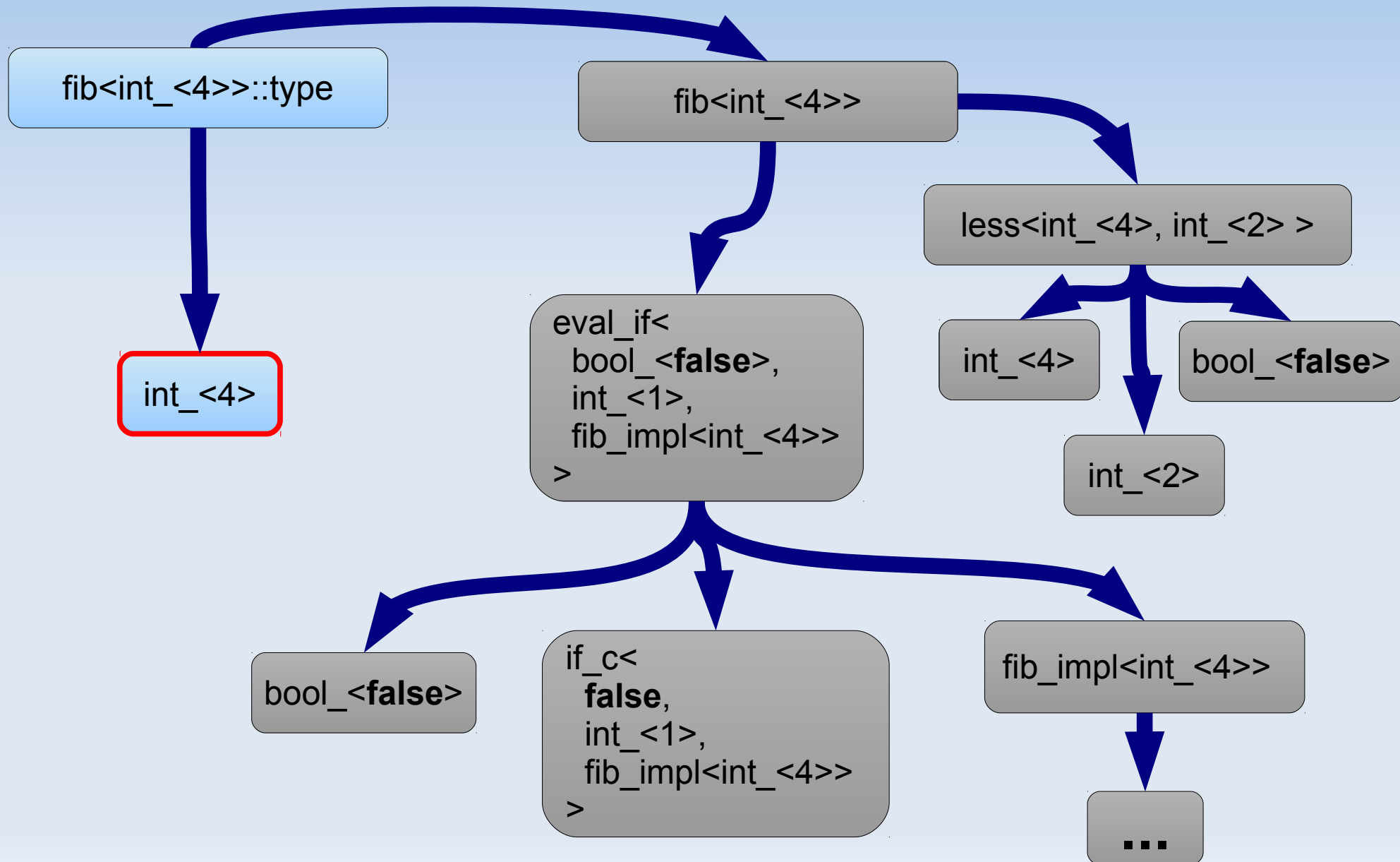
Fibonacci



Fibonacci

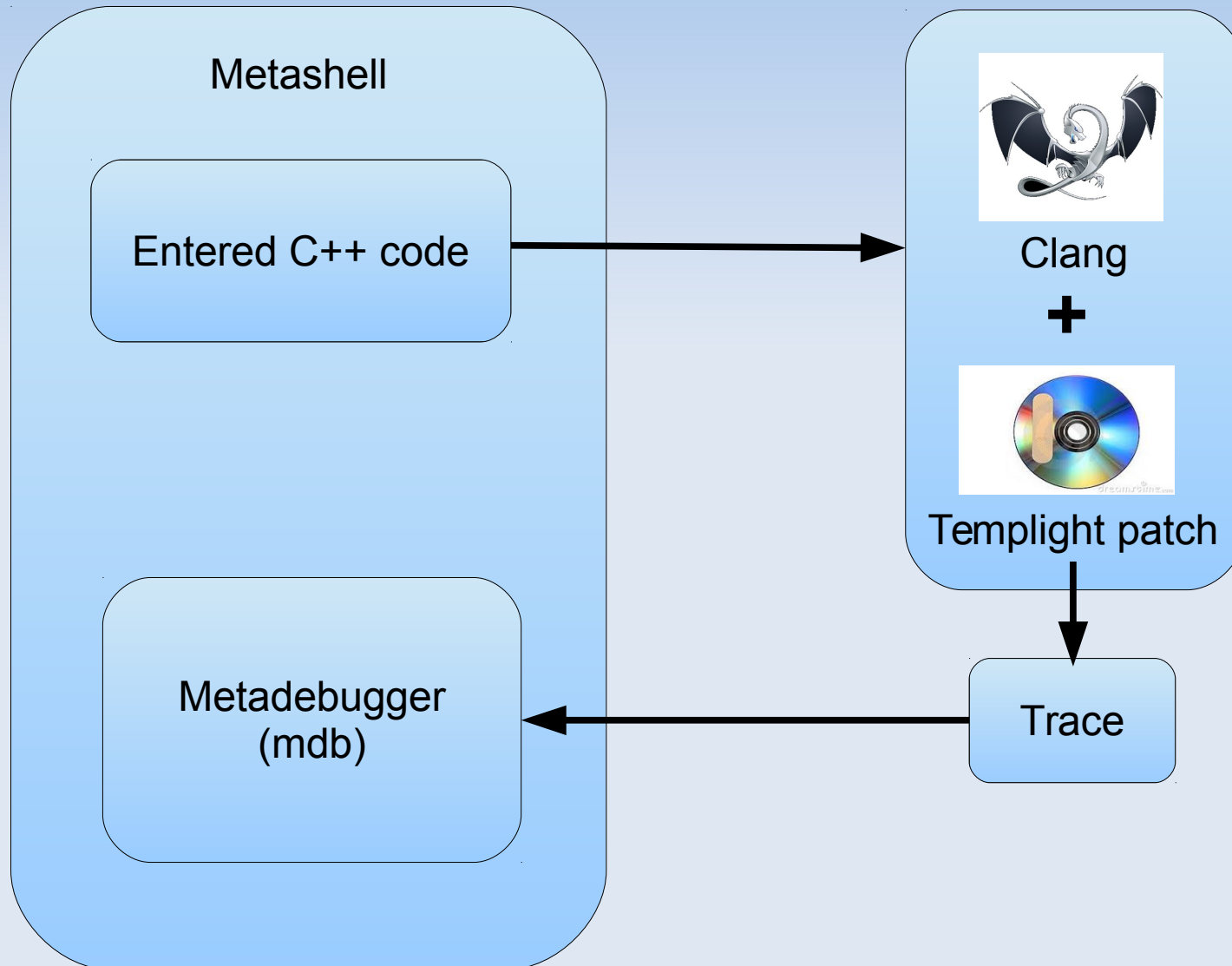


Fibonacci



Demo

Metadebugger



Demo

Tool support

Run-time



Compile-time



Tool support

Run-time



Compile-time



Future plans

- Display source location in the debugger
- Debug crashing metaprograms
- Interactive indentation of complex template instances
- Integrate Mikael Persson's Templight fork

Q & A

Ábel Sinkovics abel@sinkovics.hu
<http://github.com/sabel83>

András Kucsma andras.kucsma@gmail.com
<http://github.com/r0mai>

Metashell <http://github.com/sabel83/metashell>
Online demo <http://abel.web.elte.hu/shell>